PROGRAMACIÓN DECLARATIVA: LÓGICA Y RESTRICCIONES

Grado en Ingeniería Informática

10 de Junio de 2016 **Examen de Evaluación Continua**

Apellidos, Nombre:

Nº de matrícula:

DURACIÓN: 60 minutos

INSTRUCCIONES: El examen consta de 5 preguntas. Cada pregunta puntúa un máximo de 2 puntos. Todas las preguntas se deben contestar exclusivamente en las hojas de examen, en los espacios habilitados para tal efecto.

Pregunta 1. Escribir en ***programación lógica pura*** el predicado sumaNumPares/2 que se verifica (sumaNumPares(L,S)) si S es la suma de los números pares de la lista de números naturales L. Se dispone del predicado modulo/3 que es cierto si el primer argumento es un número par.

```
sumaNumPares([],0).
sumaNumPares([E|R],S):-
     modulo(E,s(s(0)),0),
     sumaNumPares(R,SR),
     suma(E,SR,S).
sumaNumPares([E|R],S):-
     modulo(E,s(s(0)),s(0)),
     sumaNumPares(R,S).

suma(0,X,X).
suma(s(X),Y, s(Z)):- suma(X,Y,Z).
```

Pregunta 2. Escribir en ***ISO Prolog*** el predicado listaImpares/2 (listaImpares(L,LI)) que se verifica si dada una lista de números (L), LI es la lista de los números impares de L. Se dispone de un predicado esImpar(N) que se verifica si N es un número impar. El predicado pedido debe fallar forzosamente (es decir, no debe dar error) si el primer argumento no está totalmente instanciado. Se dispone del predicado esImpar(N) que es cierto si N es un número impar.

Proporcionar (a) una versión recursiva del predicado pedido y (b) otra versión usando predicados de agregación, de manera que la recursividad no sea explícita en el predicado pedido.

(a) versión recursiva

```
listaImpares(X,[]):-
     nonvar(X),
     X = [].
listaImpares([X|Xs],[X|Ys]):-
     number(X),
     esImpar(X), !,
     listaImpares(Xs,Ys).
listaImpares([X|Xs],Ys):-
     number(X),
     listaImpares(Xs,Ys).
```

```
(b) versión no recursiva

listaImpares(L,LI):-
     ground(L),
     findall(X, ( member(X,L), number(X), esImpar(X) ), LI).
```

Pregunta 3. (*2 puntos*) Dado el siguiente programa lógico:

jefe(a1,a2).

jefe(a1,b2).

jefe(a2,a3).

jefe(a2,c3).

jefe(b2,d3).

Proporcionar todas las respuestas (una por línea) a la consulta 'es_jefe(X)', suponiendo las siguientes definiciones:

(a) es_jefe(X) :- subordinado(_Y,X).

subordinado(X,Y) :- jefe(Y,X).

```
X = a1 ? ;
X = a1 ? ;
X = a2 ? ;
X = a2 ? ;
X = b2 ? ;
no
```

(b) es_jefe(X) :- subordinado(_Y,X),!.

subordinado(X,Y) :- jefe(Y,X).

```
X = a1 ? ;
no
```

(c) es_jefe(X) :- jefe(X,Y), subordinado(Z,X), Z=Y.

subordinado(X,Y) :- jefe(Y,X),!.

```
X = a1 ? ;
X = a2 ? ;
X = b2 ? ;
no
```

Pregunta 4.

(a) Dado el siguiente programa lógico:

```
cocinaAutor(elClubAllard).
cocinaAutor(diverXo).
cocinaAutor(dassaBassa).
precioAlto(diverXo).
recomendado(Restaurante) :- \+ precioAlto(Restaurante).
```

Proporcionar todas las respuestas (una por línea) a la siguiente consulta:

```
?- recomendado(X), cocinaAutor(X).
    no
```

(b) Implementar un predicado nand/7 que simule una puerta lógica NAND (equivalente a NOT AND) de 7 entradas utilizando obligatoriamente para ello una construcción *cut-fail*.

Ejemplos de consultas:

```
?- nand(1,1,1,1,1,1,1).
no
?- nand(1,1,0,1,1,1,1).
yes
```

```
nand(1,1,1,1,1,1,1) :- !, fail.
nand(_,_,_,_,_,_,_).
```

Pregunta 5. Dado el siguiente programa lógico:

```
mult(X, Y, Z) :- Z is X * Y.
multR(X, Y, Z) :- Z .=. X * Y
```

Proporcionar todas las respuestas (una por línea) a las siguientes consultas:

```
?- mult(3, 3, Z).
        Z = 9
?- mult(X, 3, 7).
        Error
?- multR(4,3, Z).
        Z = 12
?- multR(X, 4, 1).
        X = 0.25
?- multR(X, Y, 12).
        12 .=. X * Y
```